// ==============================================================

// RTL generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC

// Version: 2017.2

// Copyright (C) 1986-2017 Xilinx, Inc. All Rights Reserved.

//

// ===========================================================

#include "simple.h"

#include "AESL\_pkg.h"

using namespace std;

namespace ap\_rtl {

const sc\_logic simple::ap\_const\_logic\_1 = sc\_dt::Log\_1;

const sc\_logic simple::ap\_const\_logic\_0 = sc\_dt::Log\_0;

const sc\_lv<7> simple::ap\_ST\_fsm\_state1 = "1";

const sc\_lv<7> simple::ap\_ST\_fsm\_state2 = "10";

const sc\_lv<7> simple::ap\_ST\_fsm\_state3 = "100";

const sc\_lv<7> simple::ap\_ST\_fsm\_state4 = "1000";

const sc\_lv<7> simple::ap\_ST\_fsm\_state5 = "10000";

const sc\_lv<7> simple::ap\_ST\_fsm\_state6 = "100000";

const sc\_lv<7> simple::ap\_ST\_fsm\_state7 = "1000000";

const sc\_lv<32> simple::ap\_const\_lv32\_0 = "00000000000000000000000000000000";

const sc\_lv<32> simple::ap\_const\_lv32\_1 = "1";

const sc\_lv<1> simple::ap\_const\_lv1\_0 = "0";

const sc\_lv<32> simple::ap\_const\_lv32\_2 = "10";

const sc\_lv<4> simple::ap\_const\_lv4\_0 = "0000";

const sc\_lv<32> simple::ap\_const\_lv32\_6 = "110";

const sc\_lv<4> simple::ap\_const\_lv4\_8 = "1000";

const sc\_lv<4> simple::ap\_const\_lv4\_1 = "1";

const sc\_lv<1> simple::ap\_const\_lv1\_1 = "1";

const bool simple::ap\_const\_boolean\_1 = true;

simple::simple(sc\_module\_name name) : sc\_module(name), mVcdFile(0) {

simple\_fadd\_32ns\_bkb\_U1 = new simple\_fadd\_32ns\_bkb<1,5,32,32,32>("simple\_fadd\_32ns\_bkb\_U1");

simple\_fadd\_32ns\_bkb\_U1->clk(ap\_clk);

simple\_fadd\_32ns\_bkb\_U1->reset(ap\_rst);

simple\_fadd\_32ns\_bkb\_U1->din0(a\_q0);

simple\_fadd\_32ns\_bkb\_U1->din1(b\_q0);

simple\_fadd\_32ns\_bkb\_U1->ce(ap\_var\_for\_const0);

simple\_fadd\_32ns\_bkb\_U1->dout(grp\_fu\_73\_p2);

SC\_METHOD(thread\_ap\_clk\_no\_reset\_);

dont\_initialize();

sensitive << ( ap\_clk.pos() );

SC\_METHOD(thread\_a\_address0);

sensitive << ( ap\_CS\_fsm\_state2 );

sensitive << ( tmp\_fu\_92\_p1 );

SC\_METHOD(thread\_a\_ce0);

sensitive << ( ap\_CS\_fsm\_state2 );

SC\_METHOD(thread\_ap\_CS\_fsm\_state1);

sensitive << ( ap\_CS\_fsm );

SC\_METHOD(thread\_ap\_CS\_fsm\_state2);

sensitive << ( ap\_CS\_fsm );

SC\_METHOD(thread\_ap\_CS\_fsm\_state3);

sensitive << ( ap\_CS\_fsm );

SC\_METHOD(thread\_ap\_CS\_fsm\_state7);

sensitive << ( ap\_CS\_fsm );

SC\_METHOD(thread\_ap\_done);

sensitive << ( ap\_CS\_fsm\_state2 );

sensitive << ( exitcond\_fu\_80\_p2 );

SC\_METHOD(thread\_ap\_idle);

sensitive << ( ap\_start );

sensitive << ( ap\_CS\_fsm\_state1 );

SC\_METHOD(thread\_ap\_ready);

sensitive << ( ap\_CS\_fsm\_state2 );

sensitive << ( exitcond\_fu\_80\_p2 );

SC\_METHOD(thread\_ap\_return);

sensitive << ( ap\_CS\_fsm\_state2 );

sensitive << ( exitcond\_fu\_80\_p2 );

SC\_METHOD(thread\_b\_address0);

sensitive << ( ap\_CS\_fsm\_state2 );

sensitive << ( tmp\_fu\_92\_p1 );

SC\_METHOD(thread\_b\_ce0);

sensitive << ( ap\_CS\_fsm\_state2 );

SC\_METHOD(thread\_c\_address0);

sensitive << ( tmp\_reg\_106 );

sensitive << ( ap\_CS\_fsm\_state7 );

SC\_METHOD(thread\_c\_ce0);

sensitive << ( ap\_CS\_fsm\_state7 );

SC\_METHOD(thread\_c\_d0);

sensitive << ( ap\_CS\_fsm\_state7 );

sensitive << ( grp\_fu\_73\_p2 );

SC\_METHOD(thread\_c\_we0);

sensitive << ( ap\_CS\_fsm\_state7 );

SC\_METHOD(thread\_exitcond\_fu\_80\_p2);

sensitive << ( ap\_CS\_fsm\_state2 );

sensitive << ( i\_reg\_62 );

SC\_METHOD(thread\_i\_1\_fu\_86\_p2);

sensitive << ( i\_reg\_62 );

SC\_METHOD(thread\_tmp\_fu\_92\_p1);

sensitive << ( i\_reg\_62 );

SC\_METHOD(thread\_ap\_NS\_fsm);

sensitive << ( ap\_start );

sensitive << ( ap\_CS\_fsm );

sensitive << ( ap\_CS\_fsm\_state1 );

sensitive << ( ap\_CS\_fsm\_state2 );

sensitive << ( exitcond\_fu\_80\_p2 );

SC\_THREAD(thread\_hdltv\_gen);

sensitive << ( ap\_clk.pos() );

SC\_THREAD(thread\_ap\_var\_for\_const0);

ap\_CS\_fsm = "0000001";

static int apTFileNum = 0;

stringstream apTFilenSS;

apTFilenSS << "simple\_sc\_trace\_" << apTFileNum ++;

string apTFn = apTFilenSS.str();

mVcdFile = sc\_create\_vcd\_trace\_file(apTFn.c\_str());

mVcdFile->set\_time\_unit(1, SC\_PS);

if (1) {

#ifdef \_\_HLS\_TRACE\_LEVEL\_PORT\_\_

sc\_trace(mVcdFile, ap\_clk, "(port)ap\_clk");

sc\_trace(mVcdFile, ap\_rst, "(port)ap\_rst");

sc\_trace(mVcdFile, ap\_start, "(port)ap\_start");

sc\_trace(mVcdFile, ap\_done, "(port)ap\_done");

sc\_trace(mVcdFile, ap\_idle, "(port)ap\_idle");

sc\_trace(mVcdFile, ap\_ready, "(port)ap\_ready");

sc\_trace(mVcdFile, a\_address0, "(port)a\_address0");

sc\_trace(mVcdFile, a\_ce0, "(port)a\_ce0");

sc\_trace(mVcdFile, a\_q0, "(port)a\_q0");

sc\_trace(mVcdFile, b\_address0, "(port)b\_address0");

sc\_trace(mVcdFile, b\_ce0, "(port)b\_ce0");

sc\_trace(mVcdFile, b\_q0, "(port)b\_q0");

sc\_trace(mVcdFile, c\_address0, "(port)c\_address0");

sc\_trace(mVcdFile, c\_ce0, "(port)c\_ce0");

sc\_trace(mVcdFile, c\_we0, "(port)c\_we0");

sc\_trace(mVcdFile, c\_d0, "(port)c\_d0");

sc\_trace(mVcdFile, ap\_return, "(port)ap\_return");

#endif

#ifdef \_\_HLS\_TRACE\_LEVEL\_INT\_\_

sc\_trace(mVcdFile, ap\_CS\_fsm, "ap\_CS\_fsm");

sc\_trace(mVcdFile, ap\_CS\_fsm\_state1, "ap\_CS\_fsm\_state1");

sc\_trace(mVcdFile, i\_1\_fu\_86\_p2, "i\_1\_fu\_86\_p2");

sc\_trace(mVcdFile, i\_1\_reg\_101, "i\_1\_reg\_101");

sc\_trace(mVcdFile, ap\_CS\_fsm\_state2, "ap\_CS\_fsm\_state2");

sc\_trace(mVcdFile, tmp\_fu\_92\_p1, "tmp\_fu\_92\_p1");

sc\_trace(mVcdFile, tmp\_reg\_106, "tmp\_reg\_106");

sc\_trace(mVcdFile, exitcond\_fu\_80\_p2, "exitcond\_fu\_80\_p2");

sc\_trace(mVcdFile, ap\_CS\_fsm\_state3, "ap\_CS\_fsm\_state3");

sc\_trace(mVcdFile, i\_reg\_62, "i\_reg\_62");

sc\_trace(mVcdFile, ap\_CS\_fsm\_state7, "ap\_CS\_fsm\_state7");

sc\_trace(mVcdFile, grp\_fu\_73\_p2, "grp\_fu\_73\_p2");

sc\_trace(mVcdFile, ap\_NS\_fsm, "ap\_NS\_fsm");

#endif

}

mHdltvinHandle.open("simple.hdltvin.dat");

mHdltvoutHandle.open("simple.hdltvout.dat");

}

simple::~simple() {

if (mVcdFile)

sc\_close\_vcd\_trace\_file(mVcdFile);

mHdltvinHandle << "] " << endl;

mHdltvoutHandle << "] " << endl;

mHdltvinHandle.close();

mHdltvoutHandle.close();

delete simple\_fadd\_32ns\_bkb\_U1;

}

void simple::thread\_ap\_var\_for\_const0() {

ap\_var\_for\_const0 = ap\_const\_logic\_1;

}

void simple::thread\_ap\_clk\_no\_reset\_() {

if ( ap\_rst.read() == ap\_const\_logic\_1) {

ap\_CS\_fsm = ap\_ST\_fsm\_state1;

} else {

ap\_CS\_fsm = ap\_NS\_fsm.read();

}

if (esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state7.read())) {

i\_reg\_62 = i\_1\_reg\_101.read();

} else if ((esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state1.read()) &&

esl\_seteq<1,1,1>(ap\_start.read(), ap\_const\_logic\_1))) {

i\_reg\_62 = ap\_const\_lv4\_0;

}

if (esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state2.read())) {

i\_1\_reg\_101 = i\_1\_fu\_86\_p2.read();

}

if ((esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state2.read()) && esl\_seteq<1,1,1>(exitcond\_fu\_80\_p2.read(), ap\_const\_lv1\_0))) {

tmp\_reg\_106 = tmp\_fu\_92\_p1.read();

}

}

void simple::thread\_a\_address0() {

a\_address0 = (sc\_lv<3>) (tmp\_fu\_92\_p1.read());

}

void simple::thread\_a\_ce0() {

if (esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state2.read())) {

a\_ce0 = ap\_const\_logic\_1;

} else {

a\_ce0 = ap\_const\_logic\_0;

}

}

void simple::thread\_ap\_CS\_fsm\_state1() {

ap\_CS\_fsm\_state1 = ap\_CS\_fsm.read()[0];

}

void simple::thread\_ap\_CS\_fsm\_state2() {

ap\_CS\_fsm\_state2 = ap\_CS\_fsm.read()[1];

}

void simple::thread\_ap\_CS\_fsm\_state3() {

ap\_CS\_fsm\_state3 = ap\_CS\_fsm.read()[2];

}

void simple::thread\_ap\_CS\_fsm\_state7() {

ap\_CS\_fsm\_state7 = ap\_CS\_fsm.read()[6];

}

void simple::thread\_ap\_done() {

if ((esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state2.read()) &&

esl\_seteq<1,1,1>(exitcond\_fu\_80\_p2.read(), ap\_const\_lv1\_1))) {

ap\_done = ap\_const\_logic\_1;

} else {

ap\_done = ap\_const\_logic\_0;

}

}

void simple::thread\_ap\_idle() {

if ((esl\_seteq<1,1,1>(ap\_const\_logic\_0, ap\_start.read()) &&

esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state1.read()))) {

ap\_idle = ap\_const\_logic\_1;

} else {

ap\_idle = ap\_const\_logic\_0;

}

}

void simple::thread\_ap\_ready() {

if ((esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state2.read()) &&

esl\_seteq<1,1,1>(exitcond\_fu\_80\_p2.read(), ap\_const\_lv1\_1))) {

ap\_ready = ap\_const\_logic\_1;

} else {

ap\_ready = ap\_const\_logic\_0;

}

}

void simple::thread\_ap\_return() {

ap\_return = ap\_const\_lv32\_0;

}

void simple::thread\_b\_address0() {

b\_address0 = (sc\_lv<3>) (tmp\_fu\_92\_p1.read());

}

void simple::thread\_b\_ce0() {

if (esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state2.read())) {

b\_ce0 = ap\_const\_logic\_1;

} else {

b\_ce0 = ap\_const\_logic\_0;

}

}

void simple::thread\_c\_address0() {

c\_address0 = (sc\_lv<3>) (tmp\_reg\_106.read());

}

void simple::thread\_c\_ce0() {

if (esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state7.read())) {

c\_ce0 = ap\_const\_logic\_1;

} else {

c\_ce0 = ap\_const\_logic\_0;

}

}

void simple::thread\_c\_d0() {

c\_d0 = grp\_fu\_73\_p2.read();

}

void simple::thread\_c\_we0() {

if (esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state7.read())) {

c\_we0 = ap\_const\_logic\_1;

} else {

c\_we0 = ap\_const\_logic\_0;

}

}

void simple::thread\_exitcond\_fu\_80\_p2() {

exitcond\_fu\_80\_p2 = (!i\_reg\_62.read().is\_01() || !ap\_const\_lv4\_8.is\_01())? sc\_lv<1>(): sc\_lv<1>(i\_reg\_62.read() == ap\_const\_lv4\_8);

}

void simple::thread\_i\_1\_fu\_86\_p2() {

i\_1\_fu\_86\_p2 = (!i\_reg\_62.read().is\_01() || !ap\_const\_lv4\_1.is\_01())? sc\_lv<4>(): (sc\_biguint<4>(i\_reg\_62.read()) + sc\_biguint<4>(ap\_const\_lv4\_1));

}

void simple::thread\_tmp\_fu\_92\_p1() {

tmp\_fu\_92\_p1 = esl\_zext<64,4>(i\_reg\_62.read());

}

void simple::thread\_ap\_NS\_fsm() {

switch (ap\_CS\_fsm.read().to\_uint64()) {

case 1 :

if ((esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state1.read()) && esl\_seteq<1,1,1>(ap\_start.read(), ap\_const\_logic\_1))) {

ap\_NS\_fsm = ap\_ST\_fsm\_state2;

} else {

ap\_NS\_fsm = ap\_ST\_fsm\_state1;

}

break;

case 2 :

if ((esl\_seteq<1,1,1>(ap\_const\_logic\_1, ap\_CS\_fsm\_state2.read()) && esl\_seteq<1,1,1>(exitcond\_fu\_80\_p2.read(), ap\_const\_lv1\_1))) {

ap\_NS\_fsm = ap\_ST\_fsm\_state1;

} else {

ap\_NS\_fsm = ap\_ST\_fsm\_state3;

}

break;

case 4 :

ap\_NS\_fsm = ap\_ST\_fsm\_state4;

break;

case 8 :

ap\_NS\_fsm = ap\_ST\_fsm\_state5;

break;

case 16 :

ap\_NS\_fsm = ap\_ST\_fsm\_state6;

break;

case 32 :

ap\_NS\_fsm = ap\_ST\_fsm\_state7;

break;

case 64 :

ap\_NS\_fsm = ap\_ST\_fsm\_state2;

break;

default :

ap\_NS\_fsm = "XXXXXXX";

break;

}

}

void simple::thread\_hdltv\_gen() {

const char\* dump\_tv = std::getenv("AP\_WRITE\_TV");

if (!(dump\_tv && string(dump\_tv) == "on")) return;

wait();

mHdltvinHandle << "[ " << endl;

mHdltvoutHandle << "[ " << endl;

int ap\_cycleNo = 0;

while (1) {

wait();

const char\* mComma = ap\_cycleNo == 0 ? " " : ", " ;

mHdltvinHandle << mComma << "{" << " \"ap\_rst\" : \"" << ap\_rst.read() << "\" ";

mHdltvinHandle << " , " << " \"ap\_start\" : \"" << ap\_start.read() << "\" ";

mHdltvoutHandle << mComma << "{" << " \"ap\_done\" : \"" << ap\_done.read() << "\" ";

mHdltvoutHandle << " , " << " \"ap\_idle\" : \"" << ap\_idle.read() << "\" ";

mHdltvoutHandle << " , " << " \"ap\_ready\" : \"" << ap\_ready.read() << "\" ";

mHdltvoutHandle << " , " << " \"a\_address0\" : \"" << a\_address0.read() << "\" ";

mHdltvoutHandle << " , " << " \"a\_ce0\" : \"" << a\_ce0.read() << "\" ";

mHdltvinHandle << " , " << " \"a\_q0\" : \"" << a\_q0.read() << "\" ";

mHdltvoutHandle << " , " << " \"b\_address0\" : \"" << b\_address0.read() << "\" ";

mHdltvoutHandle << " , " << " \"b\_ce0\" : \"" << b\_ce0.read() << "\" ";

mHdltvinHandle << " , " << " \"b\_q0\" : \"" << b\_q0.read() << "\" ";

mHdltvoutHandle << " , " << " \"c\_address0\" : \"" << c\_address0.read() << "\" ";

mHdltvoutHandle << " , " << " \"c\_ce0\" : \"" << c\_ce0.read() << "\" ";

mHdltvoutHandle << " , " << " \"c\_we0\" : \"" << c\_we0.read() << "\" ";

mHdltvoutHandle << " , " << " \"c\_d0\" : \"" << c\_d0.read() << "\" ";

mHdltvoutHandle << " , " << " \"ap\_return\" : \"" << ap\_return.read() << "\" ";

mHdltvinHandle << "}" << std::endl;

mHdltvoutHandle << "}" << std::endl;

ap\_cycleNo++;

}

}

}